SIGNAL PROCESSING ALGORITHMS - GEORGIA TECH BENCHMARK

SPECIAL TECHNICAL REPORT

REPORT NO. STR-0142-90-008

February 27, 1990

GUIDANCE, NAVIGATION AND CONTROL

DIGITAL EMULATION TECHNOLOGY LABORATORY

Contract No. DASG60-89-C-0142

Sponsored By

The United States Army Strategic Defense Command

COMPUTER ENGINEERING RESEARCH LABORATORY

Georgia Institute of Technology

Atlanta, Georgia 30332 - 0540

Contract Data Requirements List Item A008

Period Covered: Not Applicable

Type Report: As Required

DISCLAIMER

DISTRIBUTION CONTROL

SIGNAL PROCESSING ALGORITHMS -

GEORGIA TECH BENCHMARK

February 27, 1990

Authors

Andrew M. Henshaw, Steven R. Gieseking, Roy W. Melton

COMPUTER ENGINEERING RESEARCH LABORATORY

Georgia Institute of Technology

Atlanta, Georgia 30332-0540

Eugene L. Sanders	Cecil O. Alford
USASDC	Georgia Tech
Contract Monitor	Project Director

Introduction 1

Harness 1

Description 1

Module Listing 2

Non-Uniformity Compensation 4

Description 4

Data fields 4

Module Listing 4

Spatial Filtering 8

Description 8

Data Fields 8

Module Listing 8

Temporal Filtering 12

Description 12

Data Fields 12

Module Listing 12

Thresholding 15

Simple Thresholding 15

Data Fields 15

Module Listing 15

Adaptive Thresholding 18

Data Fields 18

Module Listing 18

Clustering & Centroiding 21

Description 21

Data Fields 21

Module Listing 21

I. Introduction

This document describes a set of signal-processing algorithms, as implemented by the Computer Engineering Research Laboratory at Georgia Tech. The routines are presented as a representative collection of operations for processing Infrared Focal-Plane Array signals.

For the purposes of testing and dissemination, each algorithm is presented as a stand-alone FORTRAN program. These programs are based upon a core *harness* routine which supports the input/output of a common data format (Georgia Tech Algorithm Evaluation Data Format - described in the Harness section). The modular implementations offer several benefits:

- simplification of the generation of test vectors for the verification of alternate implementations

- capability for testing various algorithm combinations, without re-compilation

- support for multiple language and/or processor-platform implementations

II. Harness

A. Description

The *Harness* program shown below is the basis of the input/output methodology used by all of the routines in this document. The code implements a simple Pass-Through module which reads a data stream, picking off the FPA pixel data, and writing the data onto an output data stream.

The Georgia Tech Algorithm Evaluation Data format is a simple ASCII text representation of a data stream. The data stream has two major components - the *Field Header* and the *Field Data*. The harness of each module processes the data stream by reading each line and checking for Field Headers which are relevant to that module. Any lines which are not relevant, or unrecognized, are immediately placed upon the output data stream. As soon as a relevant Field Header is recognized, the Field Data which follows is processed in a manner which is appropriate to that module and Field Header. This scheme provides for the chaining of modules output-to-input, without either module requiring knowledge of all, or any, of the other module's data formats. In typical use, controls for many modules could be included in a single data stream; each module would only process data intended for it. For example, suppose a test setup was composed of the following pipeline:

Input data stream ----> Spatial Filter ----> Simple Threshold ----> Output Data Stream

The data stream might appear as follows:

Input Data Stream	Description	Used by	Action
Dimensions 128	Field Header Field Data	Spatial Filter and Simple Threshold	input
Simple Thresholding Limits 0 256	Field Header Field Data	Simple Threshold	input
Pixel Data 99 93 ... 81 76	Field Header Field Data Field Data	Spatial Filter and Simple Threshold	input, modified
End	Field Header	Spatial Filter and Simple Threshold	input

Output Data Stream	Description	Generated by	Action
Dimensions 128	Field Header Field Data	Spatial Filter and Simple Threshold	copied to output data stream
Simple Thresholding Limits 0 256	Field Header Field Data	Simple Threshold	copied to output data stream
Simple Thresholding Statistics 0 256 1024	Field Header Field Data	Simple Threshold	generated, then placed on output data stream
Pixel Data 99 93 ... 81 76	Field Header Field Data Field Data	Spatial Filter and Simple Threshold	modified, then placed on output data stream
End	Field Header	Spatial Filter and Simple Threshold	copied to output data stream

B. Module Listing

```
      PROGRAM HARNESS
C
C     This program acts as a harness for testing various
C     Signal Processing Routines
C
C     Computer Engineering Research Laboratory
C     Georgia Institute of Technology
```

```
C     400 Tenth St.  CRB 390
C     Atlanta, GA  30332-0540
C     Contact: Andrew Henshaw  (404)894-2521
C
C     Written by A. M. Henshaw     Jan 23, 1990
C     Using Microsoft Fortran
C
      CHARACTER*(*) Dim, Pixels
      PARAMETER (Dim   ='Dimensions')
      PARAMETER (Pixels='Pixel Data')
      PARAMETER (maxSize=64)

      INTEGER n
      INTEGER in(maxSize,maxSize), out(maxSize, maxSize)
      CHARACTER header*72
      LOGICAL runFlag

      WRITE (6,*) '% Processed by Pass Thru module.'
      runFlag = .TRUE.
      DO WHILE (runFlag)
        READ (5,1000) header
1000    FORMAT (A72)
        IF (header.EQ.Dim) THEN
          READ (5,*) n
          WRITE (6,*) Dim
          WRITE (6,*) n
        ELSE IF (header.EQ.Pixels) THEN
          READ (5,*) ((in(row,col),col=1,n),row=1,n)

          CALL PassThru (n, in, out)

          WRITE (6,*) Pixels
          WRITE (6,*) ((out(row,col),col=1,n),row=1,n)
        ELSE IF (header.EQ.'End') THEN
          WRITE (6,*) 'End'
          runFlag = .FALSE.
        ELSE
          WRITE (6,*) header
        END IF
      END DO

      END

C*********************************************************************

      SUBROUTINE PassThru (n, in, out)

      PARAMETER (maxSize=64)
      INTEGER n, row, col
      INTEGER in(maxSize, maxSize)
      INTEGER out(maxSize, maxSize)


      DO 30 row = 1, n
        DO 30 col = 1, n
          out(row,col) = in(row,col)
30    CONTINUE

      RETURN
      END
```

III. Non-Uniformity Compensation

A. Description

The non-uniformity compensation algorithm provides a pixel-by-pixel correction of the actual pixel response to the desired response. The current algorithm uses up to a five-point, piecewise-linear correction to the pixel intensity. The correction is determined by sending a specified number of calibration frames through the process. Each of the calibration frames will have been generated by exposing the focal-plane array to a known intensity so that a desired pixel intensity is expected at each pixel.

Dead, or inadequately responsive, pixels are assumed to have been marked by another module and, during processing, they are replaced by the intensity of the previous pixel.

After calibration is performed, the algorithm enters the processing phase. For each pixel which is to be processed, it is first determined if it is a dead pixel. For normal pixels, the calibration intensities are searched to determine which section should be used for correction. After the section is determined, a linear interpolation is performed using the input pixel intensity to interpolate between the desired responses.

B. Data fields

Action	**Field Header**	**Field Data**	**Data Type**
input	*Dimensions*	FPA dimension	Integer
input	*Calibration Frames*	Count of calibration frames	Integer
input	*Calibration Input*	Vector of reference inputs	Integer [1..Count]
input	*Calibration Pixel Data*	Array of pixel response data for one input reference	Integer [1..Dimension] [1..Dimension]
modify	*Pixel Data*	Pixel data array	Integer [1..Dimension] [1..Dimension]

C. Module Listing

```
      PROGRAM NUNICOMP
C
C     Non-Uniform Compensation Test Module
C
C     Computer Engineering Research Laboratory
C     Georgia Institute of Technology
C     400 Tenth St.  CRB 390
C     Atlanta, GA  30332-0540
C     Contact: Andrew Henshaw  (404)894-2521
C
C     conforms to the Ga. Tech Algorithm Evaluation Data Format
C
C     Fortran translation of Occam code
C     Steve Gieseking
```

```
C     Roy W. Melton     Feb  1, 1990
C
C     Harness written by Andrew Henshaw     Jan 23, 1990
C     Using Microsoft Fortran
C
      CHARACTER*(*) CalInp, CalOutp, Dim, Pixels, Sect
C
C     Valid Section Headers
C
      PARAMETER (CalInp  = 'Calibration Input')
      PARAMETER (CalOutp = 'Calibration Pixel Data')
      PARAMETER (Dim     = 'Dimensions')
      PARAMETER (Pixels  = 'Pixel Data')
      PARAMETER (Sect    = 'Calibration Frames')
C
      PARAMETER (maxSize = 128)       ! maximum FPA size
      PARAMETER (maxCalFrames = 5)    ! default value

      INTEGER N, Count, Sections
      INTEGER Ic (maxCalFrames)
      INTEGER In (maxSize, maxSize), Out(maxSize, maxSize)
      INTEGER Oc (maxCalFrames, maxSize, maxSize)
      CHARACTER Header*72
      LOGICAL runFlag

      Count = 1
      Sections = maxCalFrames
      WRITE (6,*) '% Processed by Non-Uniformity Compensation Module.'
      runFlag = .TRUE.
      DO WHILE (runFlag)
        READ (5,1000) Header
1000    FORMAT (A72)
        IF (Header.EQ.CalInp) THEN
          IF (Count.LE.Sections) THEN
            READ (5,*) Ic (Count)
            WRITE (6,*) CalInp
            WRITE (6,*) Ic (Count)
          ELSE
            WRITE (6,*) CalInp
          ENDIF

        ELSEIF (Header.EQ.CalOutp) THEN
          IF (Count.LE.Sections) THEN
            READ (5, *) ((Oc (Count, Row, Col), Col=1,N), Row=1,N)
            WRITE (6,*) CalOutp
            WRITE (6,*) ((Oc (Count, Row, Col), Col=1,N), Row=1,N)
            Count = Count + 1
          ELSE
            WRITE (6,*) CalOutp
          ENDIF

        ELSEIF (Header.EQ.Dim) THEN
          READ (5,*) N
          WRITE (6,*) Dim
          WRITE (6,*) N

        ELSE IF (Header.EQ.Pixels) THEN
          IF ((Count.GT.1).AND.(Sections.GT.1)) THEN
            READ (5,*) ((In(row,col),col=1,n),row=1,n)
            IF (Count.LE.Sections) THEN
              Sections = Count - 1
            ENDIF
            CALL NonUniformityCompensation
     +              (In, Out, Oc, Ic, N, Sections)
```

```
          WRITE (6,*) Pixels
          WRITE (6,*) ((Out(row,col), col=1,n), row=1,n)
        ELSE
          WRITE (6,*) Pixels
        ENDIF

      ELSEIF (Header.EQ.Sect) THEN
        READ (5,*) Sections
        WRITE (6,*) Sect
        WRITE (6,*) Sections

      ELSE IF (Header.EQ.'End') THEN
        WRITE (6,*) 'End'
        runFlag = .FALSE.
      ELSE
        WRITE (6,*) header
      END IF
    END DO

    END

C*********************************************************************

      SUBROUTINE NonUniformityCompensation
     +              (In, Out, Oc, Ic, N, Sections)

      PARAMETER (MAXCALFRAMES=5)
      PARAMETER (MAXSIZE=64)
      INTEGER In (MAXSIZE, MAXSIZE), Out (MAXSIZE, MAXSIZE)
      INTEGER Oc (MAXCALFRAMES, MAXSIZE, MAXSIZE)
      INTEGER Ic (MAXCALFRAMES)
      INTEGER N, Sections
      INTEGER I, J, LastPixel, Section


      LastPixel = 0
      DO 20 I = 1, N
        DO 20 J = 1, N
          IF (Oc (1, I, J).EQ.65535) THEN
            Out (I, J) = LastPixel
          ELSE
            Section = 1
10          IF ((Section.LT.(Sections - 1)).AND.
     +            (In (I, J).GE.Oc ((Section + 1), I, J))) THEN
              Section = Section + 1
              GOTO 10
            ENDIF

            IF (In (I, J).LT.Oc (Section, I, J)) THEN
              Out (I, J) = Oc (Section, I, J)
            ELSE
              Out (I, J) = (In (I, J) - OC (Section, I, J)) *
     +                     (Ic (Section + 1) - Ic (Section)) /
     +                     (Oc ((Section + 1), I, J) -
     +                      Oc (Section, I, J)        ) +
     +                     Ic (Section)
            ENDIF

            IF (Out (I, J).GT.65535) THEN
              Out (I, J) = 65535
            ENDIF
            LastPixel = Out (I, J)
          ENDIF
20    CONTINUE
```

```
RETURN
END
```

IV. Spatial Filtering

A. Description

The spatial filtering algorithm performs a convolution of the image with a 3x3 coefficient mask. This implementation supports a four point symmetric mask. Separate masks are used for the edge pixels since not all of the pixels which are needed are defined. This allows more general application of boundary conditions than would be available if the undefined pixels were treated as zeros and the same mask was used.

Since the filter allows negative coefficients in the mask, it is possible to generate negative output intensities. The coding allows the intensity to be output limited to a positive range.

B. Data Fields

Action	Field Header	Field Data	Data Type
input	*Dimensions*	FPA dimension	Integer
input	*Spatial Filter Controls*	Filter coefficients (Corner coefficients)	Integer [1..4]
		Filter coefficients (Top coefficients)	Integer [1..4]
		Filter coefficients (Right coefficients)	Integer [1..4]
		Filter coefficients (Center coefficients)	Integer [1..4]
modify	*Pixel Data*	Pixel data array	Integer [1..Dimension] [1..Dimension]

C. Module Listing

```
      PROGRAM SPFILT
C
C     Spatial Filtering Test Module
C
C     Computer Engineering Research Laboratory
C     Georgia Institute of Technology
C     400 Tenth St.  CRB 390
C     Atlanta, GA  30332-0540
C     Contact: Andrew Henshaw  (404)894-2521
C
C     conforms to the Ga. Tech Algorithm Evaluation Data Format
C
C     Fortran translation of Occam code
C     Steve Gieseking
C     Roy Melton
C
C     Harness written by Andrew Henshaw     Jan 23, 1990
C     Using Microsoft Fortran
C
      CHARACTER*(*) Controls, Dim, Pixels
      PARAMETER (Controls = 'Spatial Filter Controls')
```

```
      PARAMETER (Dim       = 'Dimensions')
      PARAMETER (Pixels    = 'Pixel Data')
      PARAMETER (maxSize   = 64)
      PARAMETER (SF_CONTROL_SIZE = 4)

      INTEGER N
      INTEGER In (maxSize, maxSize), Out (maxSize, maxSize)
      INTEGER C (SF_CONTROL_SIZE, SF_CONTROL_SIZE)

      CHARACTER header*72
      LOGICAL runFlag

      WRITE (6,*) '% Processed by Spatial Filtering Module.'

      CALL DefaultFilterControls (C)
      runFlag = .TRUE.

      DO WHILE (runFlag)
        READ (5,1000) header
 1000   FORMAT (A72)
        IF (header.EQ.Controls) THEN
          READ (5,*) ((C (row, col), col=1, SF_CONTROL_SIZE),
     +                               row=1, SF_CONTROL_SIZE  )
          WRITE (6,*) Controls
          WRITE (6,*) ((C (row, col), col=1, SF_CONTROL_SIZE),
     +                                row=1, SF_CONTROL_SIZE  )
        ELSE IF (header.EQ.Dim) THEN
          READ (5,*) N
          WRITE (6,*) Dim
          WRITE (6,*) N
        ELSE IF (header.EQ.Pixels) THEN
          READ (5,*) ((In(row,col),col=1,n),row=1,n)
          CALL SpatialFilter (In, Out, C, N)
          WRITE (6,*) Pixels
          WRITE (6,*) ((Out(row,col),col=1,n),row=1,n)
        ELSE IF (header.EQ.'End') THEN
          WRITE (6,*) 'End'
          runFlag = .FALSE.
        ELSE
          WRITE (6,*) header
        END IF
      END DO

      END

C***Filter Control Defaults*******************************************
      SUBROUTINE DefaultFilterControls (Control)

      PARAMETER (SF_CONTROL_SIZE = 4)

      INTEGER Control (SF_CONTROL_SIZE, SF_CONTROL_SIZE)
      INTEGER I, J

      DO 210 I = 1, 4
        DO 200 J = 1, 3
          Control (I, J) = 0
  200   CONTINUE

        Control (I, 4) = 16384
  210 CONTINUE

      RETURN
      END
```

```
C***Spatial Filter**************************************************
      SUBROUTINE SpatialFilter (In, Out, C, N)

      PARAMETER (MAXSIZE = 64)
      PARAMETER (SF_CONTROL_SIZE = 4)

      INTEGER In (MAXSIZE, MAXSIZE), Out (MAXSIZE, MAXSIZE)
      INTEGER C (SF_CONTROL_SIZE, SF_CONTROL_SIZE)
      INTEGER N
      INTEGER I, J

      DO 100 I = 1, N
        DO 100 J = 1, N
          IF (I.EQ.1) THEN
            IF (J.EQ.1) THEN
              Out (I, J) = (In (I+1, J+1) * C (1, 1)) +
     +                     (In (I, J+1)   * C (1, 2)) +
     +                     (In (I+1, J)   * C (1, 3)) +
     +                     (In (I, J)     * C (1, 4))
            ELSEIF (J.EQ.N) THEN
              Out (I, J) = (In (I+1, J-1) * C (1, 1)) +
     +                     (In (I, J-1)   * C (1, 2)) +
     +                     (In (I+1, J)   * C (1, 3)) +
     +                     (In (I, J)     * C (1, 4))
            ELSE
              Out (I, J) = ((In (I+1, J-1) + In (I+1, J+1)) * C (2, 1))+
     +                     ((In (I, J-1)   + In (I, J+1))   * C (2, 2))+
     +                     (In (I+1, J)                     * C (2, 3))+
     +                     (In (I, J)                       * C (2, 4))
            ENDIF
          ELSEIF (I.EQ.N) THEN
            IF (J.EQ.1) THEN
              Out (I, J) = (In (I-1, J+1) * C (1, 1)) +
     +                     (In (I, J+1)   * C (1, 2)) +
     +                     (In (I-1, J)   * C (1, 3)) +
     +                     (In (I, J)     * C (1, 4))
            ELSEIF (J.EQ.N) THEN
              Out (I, J) = (In (I-1, J-1) * C (1, 1)) +
     +                     (In (I, J-1)   * C (1, 2)) +
     +                     (In (I-1, J)   * C (1, 3)) +
     +                     (In (I, J)     * C (1, 4))
            ELSE
              Out (I, J) = ((In (I-1, J-1) + In (I-1, J+1)) * C (2, 1))+
     +                     ((In (I, J-1)   + In (I, J+1))   * C (2, 2))+
     +                     (In (I-1, J)                     * C (2, 3))+
     +                     (In (I, J)                       * C (2, 4))
            ENDIF
          ELSEIF (J.EQ.1) THEN
            Out (I, J) = ((In (I-1, J+1) + In (I+1, J+1)) * C (3, 1)) +
     +                   (In (I, J+1)                     * C (3, 2)) +
     +                   ((In (I-1, J)   + In (I +1, J))  * C (3, 3)) +
     +                   (In (I, J)                       * C (3, 4))
          ELSEIF (J.EQ.N) THEN
            Out (I, J) = ((In (I-1, J-1) + In (I+1, J-1)) * C (3, 1)) +
     +                   (In (I, J-1)                     * C (3, 2)) +
     +                   ((In (I-1, J) + In (I+1, J))     * C (3, 3)) +
     +                   (In (I, J)                       * C (3, 4))
          ELSE
            Out (I, J) = ((In (I-1, J-1) + In (I+1, J-1) +
     +                     In (I-1, J+1) + In (I+1, J+1) ) * C (4, 1)) +
     +                   ((In (I, J-1)   + In (I, J+1))    * C (4, 2)) +
     +                   ((In (I-1, J)   + In (I+1, J))    * C (4, 3)) +
     +                   (In (I, J)                        * C (4, 4))
          ENDIF
```

```
        Out (I, J) = Out (I, J) / 16384
        IF (Out (I, J).LT.0) THEN
          Out (I, J) = 0
        ELSEIF (Out (I, J).GT.65535) THEN
          Out (I, J) = 65535
        ENDIF

100 CONTINUE

    RETURN
    END
```

V. Temporal Filtering

A. Description

The temporal filtering algorithm provides a pixel-by-pixel infinite impulse response (IIR) filtering of the sequence of images which are sent through the process. This implementation allows up to a fourth-order filter.

B. Data Fields

Action	Field Header	Field Data	Data Type
input	*Dimensions*	FPA dimension	Integer
input	*Temporal Filtering Limits*	Lower limits	Integer
		Upper Limits	Integer
modify	*Pixel Data*	Pixel data array	Integer [1..Dimension] [1..Dimension]

C. Module Listing

```
      PROGRAM TEMPORALFILTER
C
C     Temporal Filter Test Module
C
C     Computer Engineering Research Laboratory
C     Georgia Institute of Technology
C     400 Tenth St.  CRB 390
C     Atlanta, GA  30332-0540
C     Contact: Andrew Henshaw  (404)894-2521
C
C     conforms to the Ga. Tech Algorithm Evaluation Data Format
C
C     Fortran translation of Occam code
C     Steve Gieseking
C     Roy Melton
C
C     Harness written by A. M. Henshaw    Jan 23, 1990
C     Using Microsoft Fortran
C
      CHARACTER*(*) Dim, Pixels, Limits
C
C     Valid Section Headers
      PARAMETER (Dim   ='Dimensions')
      PARAMETER (Pixels='Pixel Data')
      PARAMETER (Limits='Temporal Filtering Limits')
C
      PARAMETER (maxSize=64)     ! maximum FPA size
      PARAMETER (TF_CONTROL_SIZE = 24)

      INTEGER n, lower, upper
      INTEGER in(maxSize,maxSize), out(maxSize, maxSize)
      INTEGER X (maxSize, maxSize, 2, 2)
      INTEGER C (TF_CONTROL_SIZE)

      CHARACTER header*72
      LOGICAL runFlag
```

```
      DATA lower /0/
      DATA upper /32767/

      WRITE (6,*) '% Processed by Temporal Filtering Module.'
      runFlag = .TRUE.
      DO WHILE (runFlag)
        READ (5,1000) header
 1000   FORMAT (A72)
        IF (header.EQ.Dim) THEN
          READ (5,*) n
          WRITE (6,*) Dim
          WRITE (6,*) n
        ELSE IF (header.EQ.Limits) THEN
          READ (5,*) lower, upper
          WRITE (6,*) Limits
          WRITE (6,*) lower, upper
        ELSE IF (header.EQ.Pixels) THEN
          READ (5,*) ((in(row,col),col=1,n),row=1,n)

          CALL CalculateFilterControls (C, Lower, Upper)

          CALL TempFilt (In, Out, X, C, N)

          WRITE (6,*) Pixels
          WRITE (6,*) ((out(row,col),col=1,n),row=1,n)
        ELSE IF (header.EQ.'End') THEN
          WRITE (6,*) 'End'
          runFlag = .FALSE.
        ELSE
          WRITE (6,*) header
        END IF
      END DO

      END

C***Filter Control Calculation****************************************
      SUBROUTINE CalculateFilterControls (Control, Lower, Upper)

      PARAMETER (TF_CONTROL_SIZE = 24)

      INTEGER Control (TF_CONTROL_SIZE)
      INTEGER Lower, Upper
      INTEGER I, J

      DO 110 I = 0, 12, 12
        DO 100 J = 1, 8
          Control (J + I) = 1
  100   CONTINUE

        Control (9 + I) = 3
        Control (10 + I) = 1
        Control (11 + I) = Upper
        Control (12 + I) = Lower
  110 CONTINUE

      RETURN
      END

C***Temporal Filter***************************************************
      SUBROUTINE TempFilt (In, Out, X, C, N)

      PARAMETER (MAXSIZE = 64)
      PARAMETER (TF_A0 = 1)
      PARAMETER (TF_A1 = 2)
```

```
      PARAMETER (TF_A2 = 3)
      PARAMETER (TF_B0 = 4)
      PARAMETER (TF_B1 = 5)
      PARAMETER (TF_B2 = 6)
      PARAMETER (TF_SCALE_STATE = 7)
      PARAMETER (TF_SCALE_OUTPUT = 8)
      PARAMETER (TF_UPPER_LIMIT_STATE = 9)
      PARAMETER (TF_LOWER_LIMIT_STATE = 10)
      PARAMETER (TF_UPPER_LIMIT_OUTPUT = 11)
      PARAMETER (TF_LOWER_LIMIT_OUTPUT = 12)
      PARAMETER (TF_CONTROL_SIZE = 24)

      INTEGER In (MAXSIZE, MAXSIZE), Out (MAXSIZE, MAXSIZE)
      INTEGER X (MAXSIZE, MAXSIZE, 2, 2)
      INTEGER C (TF_CONTROL_SIZE)
      INTEGER N
      INTEGER I, J, K, L, Ptr, Value, XNew, YNew

      DO 10 I = 1, MAXSIZE
        DO 10 J = 1, MAXSIZE
          DO 10 K = 1, 2
            DO 10 L = 1, 2
              X (I, J, K, L) = 0
   10 CONTINUE

      DO 30 I = 1, N
        DO 30 J = 1, N
          Value = In (I, J)
          DO 20 K = 1,2
            Ptr = (K - 1) * 12
            XNew = ((C (Ptr + TF_A0) * Value) +
     +               (C (Ptr + TF_A1) * X (I, J, K, 1)) +
     +               (C (Ptr + TF_A2) * X (I, J, K, 2))    ) /
     +              C (Ptr + TF_SCALE_STATE)
            YNew = ((C (Ptr + TF_B0) * Value) +
     +               (C (Ptr + TF_B1) * X (I, J, K, 1)) +
     +               (C (Ptr + TF_B2) * X (I, J, K, 2))    ) /
     +              C (Ptr + TF_SCALE_OUTPUT)

            X (I, J, K, 2) = X (I, J, K, 1)

            IF (XNew.GT.C (Ptr + TF_UPPER_LIMIT_STATE)) THEN
              X (I, J, K, 1) = C (Ptr + TF_UPPER_LIMIT_STATE)
            ELSEIF (XNew.LT.C (Ptr + TF_LOWER_LIMIT_STATE)) THEN
              X (I, J, K, 1) = C (Ptr + TF_LOWER_LIMIT_STATE)
            ELSE
              X (I, J, K, 1) = XNew
            ENDIF

            IF (YNew.GT.C (Ptr + TF_UPPER_LIMIT_OUTPUT)) THEN
              Value = C (Ptr + TF_UPPER_LIMIT_OUTPUT)
            ELSEIF (YNew.LT.C (Ptr + TF_LOWER_LIMIT_OUTPUT)) THEN
              Value = C (Ptr + TF_LOWER_LIMIT_OUTPUT)
            ELSE
              Value = YNew
            ENDIF
   20     CONTINUE

          Out (I, J) = Value
   30 CONTINUE

      RETURN
      END
```

VI. Thresholding

The thresholding algorithm is used to partition the image into points which are of interest and those that are not of interest. Pixels are zeroed if they are not of interest. A pixel is passed if the intensity is above a calculated lower threshold value and below a fixed upper threshold value. The lower threshold supports two of the modes which are in the Georgia Tech VLSI design. This includes a simple, fixed threshold and an adaptive threshold based on the average and first central absolute moment of the surrounding eight pixels.

VII. Simple Thresholding

1. Data Fields

Action	Field Header	Field Data	Data Type
input	*Dimensions*	FPA dimension	Integer
input	*Simple Thresholding Limits*	Lower limit	Integer
		Upper limit	Integer
output	*Simple Thresholding Statistics*	Lower limit used	Integer
		Upper limit used	Integer
		Count of pixels exceeding limit	Integer
modify	*Pixel Data*	Pixel data array	Integer [1..Dimension] [1..Dimension]

2. Module Listing

```
      PROGRAM STHRESH
C
C     Simple Thresholding Test Module
C
C     conforms to the Ga. Tech Algorithm Evaluation Data Format
C
C     Fortran translation of Occam code
C     Steve Gieseking
C     Andrew Henshaw
C
C     Harness written by Andrew Henshaw     Jan 23, 1990
C     Using Microsoft Fortran
C     Computer Engineering Research Laboratory
C     Georgia Institute of Technology
C
      CHARACTER*(*) Dim, Pixels, Limits
C
C     Valid Section Headers
C
      PARAMETER (Dim   ='Dimensions')
      PARAMETER (Pixels='Pixel Data')
      PARAMETER (Limits='Simple Thresholding Limits')
```

```
C
      PARAMETER (maxSize=128)      ! maximum FPA size

      INTEGER n, count, lower, upper
      INTEGER in(maxSize,maxSize), out(maxSize, maxSize)
      CHARACTER header*72
      LOGICAL runFlag
      DATA lower /0/               ! default values
      DATA upper /32767/

      WRITE (6,*) '% Processed by Simple Thresholding module.'
      runFlag = .TRUE.
      DO WHILE (runFlag)
        READ (5,1000) header
1000    FORMAT (A72)
        IF (header.EQ.Dim) THEN
          READ (5,*) n
          WRITE (6,*) Dim
          WRITE (6,*) n
        ELSE IF (header.EQ.Limits) THEN
          READ (5,*) lower, upper
          WRITE (6,*) Limits
          WRITE (6,*) lower, upper
        ELSE IF (header.EQ.Pixels) THEN
          READ (5,*) ((in(row,col),col=1,n),row=1,n)

          CALL SmpThrsh (n, lower, upper, count, in, out)

          WRITE (6,*) Pixels
          WRITE (6,*) ((out(row,col),col=1,n),row=1,n)
        ELSE IF (header.EQ.'End') THEN
          WRITE (6,*) 'End'
          runFlag = .FALSE.
        ELSE
          WRITE (6,*) header
        END IF
      END DO

      END
C*******************************************************************

      SUBROUTINE SmpThrsh (n, lower, upper, count, in, out)

      PARAMETER (maxSize=64)
      INTEGER n, lower, upper, count
      INTEGER in(maxSize, maxSize)
      INTEGER out(maxSize, maxSize)
      INTEGER row, col, pixel


      count = 0
      DO 30 row = 1, n
        DO 30 col = 1, n
          pixel = in(row,col)
          IF ((pixel.GE.lower).AND.(pixel.LE.upper)) THEN
            count = count + 1
            out(row,col) = pixel
          ELSE
            out(row,col) = 0
          END  IF
30    CONTINUE

C     Put Statistics onto data stream
```

```
WRITE (6,*) 'Simple Thrésholding Statistics'
WRITE (6,*) lower, upper, count

RETURN
END
```

VIII. Adaptive Thresholding

1. Data Fields

Action	Field Header	Field Data	Data Type
input	*Dimensions*	FPA dimension	Integer
input	*Adaptive Thresholding Parameters*	Upper limit	Integer
		k1	Integer
		k2	Integer
		k3	Integer
		Scale	Integer
output	*Adaptive Thresholding Statistics*	Upper limit used	Integer
		Count of pixels exceeding limit	Integer
modify	*Pixel Data*	Pixel data array	Integer [1..Dimension] [1..Dimension]

2. Module Listing

```
      PROGRAM ADTHRESH
C
C     Adaptive Thresholding Test Module
C
C     Computer Engineering Research Laboratory
C     Georgia Institute of Technology
C     400 Tenth St.  CRB 390
C     Atlanta, GA  30332-0540
C     Contact: Andrew Henshaw  (404)894-2521
C
C     conforms to the Ga. Tech Algorithm Evaluation Data Format
C
C     Fortran translation of Occam code
C     Steve Gieseking
C     Roy Melton

C     Harness written by A. M. Henshaw     Jan 23, 1990
C     Using Microsoft Fortran
C
      CHARACTER*(*) Dim, Pixels
      PARAMETER (Dim   ='Dimensions      ')
      PARAMETER (Pixels='Pixel Data      ')
      PARAMETER (Parms='Adaptive Thresholding Parameters')
      PARAMETER (maxSize=64)

      INTEGER n, count, k1, k2, k3, scale, sum, upper
      INTEGER in(maxSize,maxSize), out(maxSize, maxSize)
      CHARACTER header*72
      LOGICAL runFlag
      DATA k1 /1/
      DATA k2 /0/
      DATA k3 /0/
      DATA scale /8/
```

```
      DATA upper /32767/

      WRITE (6,*) '% Processed by Adaptive Thresholding module.'
      runFlag = .TRUE.
      DO WHILE (runFlag)
        READ (5,1000) header
1000    FORMAT (A72)
        IF (header.EQ.Dim) THEN
          READ (5,*) n
          WRITE (6,*) Dim
          WRITE (6,*) n
        ELSE IF (header.EQ.Parms) THEN
          READ (5,*) upper, k1, k2, k3, scale
          WRITE (6,*) Parms
          WRITE (6,*) upper, k1, k2, k3, scale
        ELSE IF (header.EQ.Pixels) THEN
          READ (5,*) ((in(row,col),col=1,n),row=1,n)

          CALL AdThrsh (n, upper, count, sum,
     +                  k1, k2, k3, scale, in, out)

          WRITE (6,*) Pixels
          WRITE (6,*) ((out(row,col),col=1,n),row=1,n)
        ELSE IF (header.EQ.'End') THEN
          WRITE (6,*) 'End'
          runFlag = .FALSE.
        ELSE
          WRITE (6,*) header
        END IF
      END DO

      END

C*******************************************************************

      SUBROUTINE AdThrsh (N, Upper, Count, Sum, K1, K2, K3, Scale,
     +                    In, Out)

      PARAMETER (maxSize=64)
      INTEGER N, Upper, Count, Sum, K1, K2, K3, Scale
      INTEGER In(maxSize, maxSize)
      INTEGER Out(maxSize, maxSize)
      INTEGER Average, I, J, K, L, Lower, Stat


      Count = 0
      Sum = 0

      DO 30 I = 1, N
        DO 30 J = 1, N
          IF (((I.EQ.1).OR.(I.EQ.N)).OR.((J.EQ.1).OR.(J.EQ.N))) THEN
            Out (I, J) = 0
          ELSE
            Average = In (I-1, J-1) + In (I-1, J) + In (I-1, J+1) +
     +                In (I, J-1)   + In (I, J+1) +
     +                In (I+1, J-1) + In (I+1, J) + In (I+1, J+1)
            Stat = 0
            DO 10 K = -1, 1
              DO 10 L = -1, 1
                IF ((K.NE.0).AND.(L.NE.0)) THEN
                  Stat = Stat + ABS ((In (I+K, J+L) * 8) - Average)
                ENDIF
   10       CONTINUE
            Lower = ((Average * K1) + (Stat * K2) + K3) / Scale
```

```
          Sum = Sum + Stat

          IF ((In (I, J).GE.Lower).AND.(In (I, J).LE.Upper)) THEN
            Out (I, J) = In (I, J)
            Count = Count + 1
          ELSE
            Out (I, J) = 0
          ENDIF
        ENDIF
   30 CONTINUE

C     Put Statistics onto data stream
      WRITE (6,*) 'Adaptive Thresholding Statistics'
      WRITE (6,*) Upper, Count

      RETURN
      END
```

IX. Clustering & Centroiding

A. Description

The clustering algorithm forms connected sets of pixels based on the surrounding pixels. Two pixels are elements of the same cluster of pixels if they are one of the eight nearest neighbors of each other. The centroiding algorithm calculates the area centroid and the intensity weighted centroid of the clusters specified by the clustering algorithm.

B. Data Fields

Action	Field Header	Field Data	Data Type
input	*Dimensions*	FPA dimension	Integer
input	*Pixel Data*	Pixel data array	Integer [1..Dimension] [1..Dimension]
output	*Clusters*	Cluster count	Integer
output	*Centroids*	Vector of the following repeated Cluster count times	
		Area centroid (X)	Integer
		Area centroid (Y)	Integer
		Intensity centroid (X)	Integer
		Intensity centroid (Y)	Integer
		Area in pixels	Integer
		Total cluster intensity	Integer

C. Module Listing

```
      PROGRAM CENTROID
C
C     Clustering and Centroiding Test Module
C
C     Computer Engineering Research Laboratory
C     Georgia Institute of Technology
C     400 Tenth St.  CRB 390
C     Atlanta, GA  30332-0540
C     Contact: Andrew Henshaw  (404)894-2521
C
C     conforms to the Ga. Tech Algorithm Evaluation Data Format
C
C     Fortran translation of Occam code
C     Steve Gieseking
C     Roy W. Melton     Feb  12, 1990
C
C     Harness written by Andrew Henshaw     Jan 23, 1990
C     Using Microsoft Fortran
C
      CHARACTER*(*) Dim, Pixels
      PARAMETER (Dim   ='Dimensions      ')
      PARAMETER (Pixels='Pixel Data      ')
      PARAMETER (MAX_SIZE=64)
      PARAMETER (MAX_CLUSTERS=1024)
```

```
      INTEGER ClusterCount, N
      INTEGER Frame (MAX_SIZE, MAX_SIZE)
      INTEGER Clusters (MAX_CLUSTERS, 6)
      CHARACTER header*72
      LOGICAL runFlag

      WRITE (6,*) '% Processed by Centroid Image Module.'
      runFlag = .TRUE.
      DO WHILE (runFlag)
        READ (5,1000) header
1000    FORMAT (A72)
        IF (header.EQ.Dim) THEN
          READ (5,*) N
          WRITE (6,*) Dim
          WRITE (6,*) N
        ELSE IF (header.EQ.Pixels) THEN
          READ (5,*) ((Frame(row,col),col=1,N),row=1,N)

          CALL CentroidImage (Frame, Clusters, N, ClusterCount)

          WRITE (6,*) Pixels
          WRITE (6,*) ((Frame(row,col),col=1,N),row=1,N)
          WRITE (6,*) 'Clusters'
          WRITE (6,*) ClusterCount
          IF (ClusterCount.GT.0) THEN
            WRITE (6,*) 'Centroids'
            WRITE (6,*) ((Clusters (row, col), col=1,6),
     +                                  row=1,ClusterCount)
          ENDIF
        ELSE IF (header.EQ.'End') THEN
          WRITE (6,*) 'End'
          runFlag = .FALSE.
        ELSE
          WRITE (6,*) header
        END IF
      END DO

      END

C*********************************************************************

      SUBROUTINE CentroidImage (Frame, CData, N, ClusterCount)

      PARAMETER (MAX_SIZE=64)
      PARAMETER (MAX_CLUSTERS=1024)
      PARAMETER (CSum = 1)
      PARAMETER (CSumX = 2)
      PARAMETER (CSumY = 3)
      PARAMETER (ISum = 4)
      PARAMETER (ISumX = 5)
      PARAMETER (ISumY = 6)
      PARAMETER (ACoorX = 1)
      PARAMETER (ACoorY = 2)
      PARAMETER (ICoorX = 3)
      PARAMETER (ICoorY = 4)
      PARAMETER (Area = 5)
      PARAMETER (Intensity = 6)
      INTEGER Frame (MAX_SIZE, MAX_SIZE)
      INTEGER CData (MAX_CLUSTERS, 6)
      INTEGER N, ClusterCount
      INTEGER C0, C1, CNM1, CN, CNP1, FinalCluster, I, J
      INTEGER Cluster (MAX_SIZE + 1), Temp (6)
      INTEGER Reassign
```

```
      DIMENSION Reassign (0:MAX_CLUSTERS-1)

      DO 10 I=1,N+1
        Cluster (I) = 0
   10 CONTINUE
      FinalCluster = 0
      Reassign (0) = 0

      DO 50 I=1,N
C       /* Initialize Row */
        C1 = 0
        CNP1 = 0
        CN = Cluster (1)
   20   IF (CN.NE.Reassign (CN)) THEN
          CN = Reassign (CN)
          GOTO 20
        ENDIF
        DO 40 J = 1, N
          CNM1 = Cluster (J+1)
   30     IF (CNM1.NE.Reassign (CNM1)) THEN
            CNM1 = Reassign (CNM1)
            GOTO 30
          ENDIF
          IF (Frame (I, J).EQ.0) THEN
            C0 = 0
          ELSEIF (C1.NE.0) THEN
C           /* Add Pixel to Cluster */
            IF ((CNM1.NE.0).AND.(CNM1.NE.C1)) THEN
C             /* Merge C1, CNM1 */
              CData (C1, CSum) = CData (C1, CSum) +
     +                           CData (CNM1, CSum) + 1
              CData (C1, CSumX) = CData (C1, CSumX) +
     +                            CData (CNM1, CSumX) + J
              CData (C1, CSumY) = CData (C1, CSumY) +
     +                            CData (CNM1, CSumY) + I
              CData (C1, ISum) = CData (C1, ISum) +
     +                           CData (CNM1, ISum) + Frame (I, J)
              CData (C1, ISumX) = CData (C1, ISumX) +
     +                            CData (CNM1, ISumX) +
     +                            (Frame (I, J) * J)
              CData (C1, ISumY) = CData (C1, ISumY) +
     +                            CData (CNM1, ISumY) +
     +                            (Frame (I, J) * I)
              CData (CNM1, CSum) = 0
              Reassign (CNM1) = C1
              CNM1 = C1
              CN = Reassign (CN)
              C0 = C1

            ELSE
C             /* Add to C1 */
              CData (C1, CSum) = CData (C1, CSum) + 1
              CData (C1, CSumX) = CData (C1, CSumX) + J
              CData (C1, CSumY) = CData (C1, CSumY) + I
              CData (C1, ISum) = CData (C1, ISum) + Frame (I, J)
              CData (C1, ISumX) = CData (C1, ISumX) +
     +                            (Frame (I, J) * J)
              CData (C1, ISumY) = CData (C1, ISumY) +
     +                            (Frame (I, J) * I)
              C0 = C1
            ENDIF

          ELSEIF (CN.NE.0) THEN
C           /* Add to CN */
```

```
            CData (CN, CSum) = CData (CN, CSum) + 1
            CData (CN, CSumX) = CData (CN, CSumX) + J
            CData (CN, CSumY) = CData (CN, CSumY) + I
            CData (CN, ISum) = CData (CN, ISum) + Frame (I, J)
            CData (CN, ISumX) = CData (CN, ISumX) + (Frame (I, J) * J)
            CData (CN, ISumY) = CData (CN, ISumY) + (Frame (I, J) * I)
            C0 = CN

          ELSEIF (CNM1.NE.0) THEN
            IF ((CNP1.NE.0).AND.(CNP1.NE.CNM1)) THEN
C             /* Merge CNM1, CNP1 */
              CData (CNM1, CSum) = CData (CNM1, CSum) +
     +                             CData (CNP1, CSum) + 1
              CData (CNM1, CSumX) = CData (CNM1, CSumX) +
     +                              CData (CNP1, CSumX) + J
              CData (CNM1, CSumY) = CData (CNM1, CSumY) +
     +                              CData (CNP1, CSumY) + I
              CData (CNM1, ISum) = CData (CNM1, ISum) +
     +                             CData (CNP1, ISum) + Frame (I, J)
              CData (CNM1, ISumX) = CData (CNM1, ISumX) +
     +                              CData (CNP1, ISumX) +
     +                              (Frame (I, J) * J)
              CData (CNM1, ISumY) = CData (CNM1, ISumY) +
     +                              CData (CNP1, ISumY) +
     +                              (Frame (I, J) * I)
              CData (CNP1, CSum) = 0
              Reassign (CNP1) = CNM1
              C0 = CNM1

            ELSE
C             /* Add to CNM1 */
              CData (CNM1, CSum) = CData (CNM1, CSum) + 1
              CData (CNM1, CSumX) = CData (CNM1, CSumX) + J
              CData (CNM1, CSumY) = CData (CNM1, CSumY) + I
              CData (CNM1, ISum) = CData (CNM1, ISum) + Frame (I, J)
              CData (CNM1, ISumX) = CData (CNM1, ISumX) +
     +                              (Frame (I, J) * J)
              CData (CNM1, ISumY) = CData (CNM1, ISumY) +
     +                              (Frame (I, J) * I)
              C0 = CNM1
            ENDIF

          ELSEIF (CNP1.NE.0) THEN
C           /* Add to CNP1 */
            CData (CNP1, CSum) = CData (CNP1, CSum) + 1
            CData (CNP1, CSumX) = CData (CNP1, CSumX) + J
            CData (CNP1, CSumY) = CData (CNP1, CSumY) + I
            CData (CNP1, ISum) = CData (CNP1, ISum) + Frame (I, J)
            CData (CNP1, ISumX) = CData (CNP1, ISumX) +
     +                            (Frame (I, J) * J)
            CData (CNP1, ISumY) = CData (CNP1, ISumY) +
     +                            (Frame (I, J) * I)
            C0 = CNP1

          ELSE
C           /* New Cluster */
            FinalCluster = FinalCluster + 1
            C0 = FinalCluster
            CData (C0, CSum) = 1
            CData (C0, CSumX) = J
            CData (C0, CSumY) = I
            CData (C0, ISum) = Frame (I, J)
            CData (C0, ISumX) = Frame (I, J) * J
            CData (C0, ISumY) = Frame (I, J) * I
```

```
          Reassign (C0) = C0
        ENDIF

        Cluster (J) = C0

C       /* Update for next column */
        C1 = C0
        CNP1 = CN
        CN = CNM1

   40   CONTINUE
   50 CONTINUE

C     /* Output Centroids */
      ClusterCount = 0
      DO 70 I=1,FinalCluster
        IF (CData (I, CSum).NE.0) THEN
C         /* Valid Cluster */
          Temp (ACoorX) = (CData (I, CSumX) +
     +                ISHFT (CData (I, CSum), -1) ) / CData (I, CSum)
          Temp (ACoorY) = (CData (I, CSumY) +
     +                ISHFT (CData (I, CSum), -1) ) / CData (I, CSum)
          Temp (ICoorX) = (CData (I, ISumX) +
     +                ISHFT (CData (I, ISum), -1) ) / CData (I, ISum)
          Temp (ICoorY) = (CData (I, ISumY) +
     +                ISHFT (CData (I, ISum), -1) ) / CData (I, ISum)
          Temp (Area) = CData (I, CSum)
          Temp (Intensity) = CData (I, ISum)
          ClusterCount = ClusterCount + 1
          DO 60 J=1,6
            CData (ClusterCount, J) = Temp (J)
   60     CONTINUE
        ENDIF

   70 CONTINUE

      RETURN
      END
```